

Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II tahun 2023/2024

Penyelesaian Permainan Word Ladder Menggunakan Algoritma UCS, Greedy Best First Search, dan A*



Disusun oleh:

Rayhan Ridhar Rahman (13522160)

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG 2024**

DAFTAR ISI

DAFTAR ISI	ii
BAB 1 DESKRIPSI TUGAS	1
BAB 2 TEORI SINGKAT	2
2.1 Algoritma Uniform-Cost Search	2
2.2 Algoritma Greedy Best First Search	2
2.3 Algoritma A*	2
BAB 3 ANALISIS DAN IMPLEMENTASI ALGORITMA	3
3.1 Analisis Pemecahan Permasalahan	3
3.1.1 Nilai Evaluasi Algoritma Uniform-Cost Search	4
3.1.2 Nilai Evaluasi Algoritma Greedy Best First Search	5
3.1.3 Nilai Evaluasi Algoritma A*	6
3.2 Source Code	6
3.2.1 englishWords.java	6
3.2.2 treeNode.java	7
3.2.3 solver.java	8
3.2.4 UCS.java	9
3.2.5 GBFS.java	10
3.2.6 Source Code A*	12
3.2.7 wordladder.java	13
BAB 4 UJI COBA	16
4.1 Tata Cara Penggunaan Program	16
4.2 Hasil Uji Coba	16
4.3 Analisis Uji Coba	19
BAB 5 PENUTUP	21
5.1 Kesimpulan	21
5.2 Saran	21
DAFTAR PUSTAKA	22
LAMPIRAN	23

BAB 1 DESKRIPSI TUGAS

Word ladder (juga dikenal sebagai Doublets, word-links, change-the-word puzzles, paragrams, laddergrams, atau word golf) adalah salah satu permainan kata yang terkenal bagi seluruh kalangan. Word ladder ditemukan oleh Lewis Carroll, seorang penulis dan matematikawan, pada tahun 1877. Pada permainan ini, pemain diberikan dua kata yang disebut sebagai start word dan end word. Untuk memenangkan permainan, pemain harus menemukan rantai kata yang dapat menghubungkan antara start word dan end word. Banyaknya huruf pada start word dan end word selalu sama. Tiap kata yang berdekatan dalam rantai kata tersebut hanya boleh berbeda satu huruf saja. Pada permainan ini, diharapkan solusi optimal, yaitu solusi yang meminimalkan banyaknya kata yang dimasukkan pada rantai kata. Berikut adalah ilustrasi serta aturan permainan.

How To Play

×

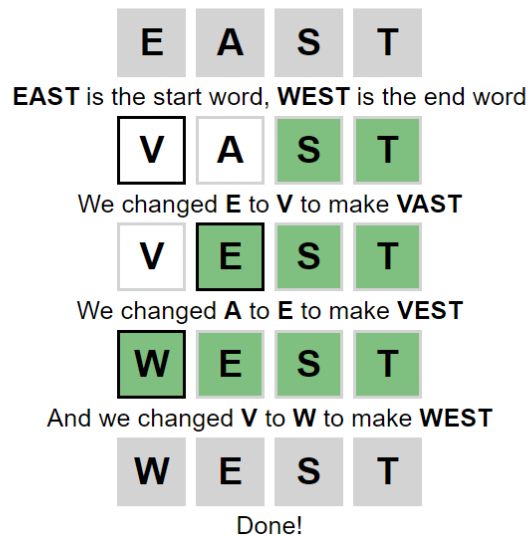
This game is called a "word ladder" and was invented by Lewis Carroll in 1877.

Rules

Weave your way from the start word to the end word.

Each word you enter **can only change 1 letter** from the word above it.

Example



Gambar 1. Ilustrasi dan Peraturan Permainan Word Ladder

(Sumber: <https://wordwormdormdork.com/>)

BAB 2 TEORI SINGKAT

2.1 Algoritma Uniform-Cost Search

BFS dan IDS dapat digunakan untuk mencari solusi dengan jumlah node yang terlewati paling minimum, bukan jarak terpendek. Algoritma uniform-cost search ini akan menelusuri graf dari akar sampai tujuan dengan mengamati jarak yang telah ditempuh sampai node tersebut. Pada awalnya algoritma ini akan terus mencari pada graf node selanjutnya melalui node aktif dengan jarak terpendek, kemudian ketika mendapatkan solusi, solusi tersebut disimpan sementara. Lalu akan dilakukan eksplorasi lagi terhadap node-node aktif yang belum ditelusuri. Jika jarak yang ditempuh melewati jarak solusi sementara, maka node tersebut menjadi dead end. Jika menemukan solusi lain dengan jarak lebih rendah, solusi sementara diganti dengan solusi tersebut. Hal ini dilakukan sampai tidak terdapat lagi node yang dapat ditelusuri. Sehingga bisa dipastikan algoritma ini dapat mencari solusi dengan jarak terendah. Tetapi karena hal tersebut juga, algoritma ini memiliki kompleksitas waktu $O(b^d)$ dan kompleksitas ruang $O(b^d)$ dengan b sebagai banyak cabang dari suatu akar dan d adalah kedalaman solusi.

2.2 Algoritma Greedy Best First Search

Algoritma ini termasuk jenis algoritma heuristik yang mengetahui nilai node berdasarkan jaraknya terhadap tujuan. Algoritma ini pada awalnya akan menelusuri node akar. Kemudian dari node akar tersebut, akan dilanjutkan dengan node akar aktif baru dengan jarak ke tujuan yang paling rendah. Jarak ke tujuan ini merupakan hasil dari perhitungan tertentu, pada contohnya dalam peta adalah jarak garis lurus ke tujuan. Kemudian akan menelusuri lagi node dengan nilai terendah. Dilakukan sampai menemukan tujuannya. Algoritma ini memiliki kompleksitas waktu $O(b^d)$ dan kompleksitas ruang $O(b^d)$ dengan b sebagai banyak cabang dari suatu akar, d adalah kedalaman terendah. Dalam kasus rata-rata, algoritma ini bisa lebih cepat daripada UCS apalagi dengan fungsi heuristic yang baik. Tetapi algoritma ini mungkin tidak mengembalikan solusi paling optimal dan dapat terjebak dalam minimal local.

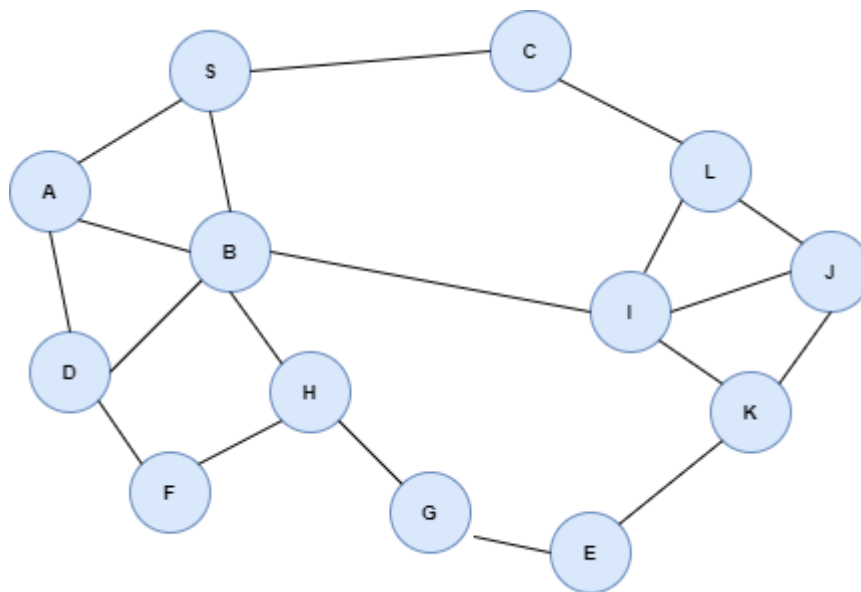
2.3 Algoritma A*

Algoritma ini merupakan gabungan dari kedua algoritma sebelumnya. Setiap node memiliki nilai jarak yang ditempuh dari awal ditambah dengan jarak untuk pergi ke tujuan. Kemudian sama dengan kedua algoritma sebelumnya ini, akan memprioritaskan untuk menelusuri node aktif dengan nilai terendah. Ketika menemukan tujuannya, maka akan langsung berhenti menelusuri graf. Dalam teori, algoritma ini komplet dengan solusi yang paling optimal. Memiliki kompleksitas waktu $O(b^m)$ dan kompleksitas ruang $O(b^m)$ dengan b sebagai banyak cabang dari suatu akar, m adalah kedalaman terendah. Algoritma ini seharusnya lebih cepat dibandingkan UCS karena akan mencoba untuk langsung menelusuri node terdekat dengan tujuan.

BAB 3 ANALISIS DAN IMPLEMENTASI ALGORITMA

3.1 Analisis Pemecahan Permasalahan

Ketiga algoritma yang akan digunakan merupakan cara untuk menentukan rute dari suatu titik ke titik lainnya dalam suatu graf. Algoritma UCS bersifat *uninformed* yang berarti algoritma tersebut akan menjelajahi graf tanpa mengetahui seberapa jauh laginya tujuan. Sedangkan kedua algoritma yang lainnya bersifat *heuristic* yang memiliki fungsi evaluasi yang dapat ditentukan, dalam algoritma yang baik akan memberi nilai estimasi yang kurang dibanding jarak ke tujuan sesungguhnya. Pada analisis permasalahan perancangan algoritma, akan digunakan graf sebagai berikut.



Gambar 3.1.1 Contoh Graf yang Akan Ditelusuri

(Sumber: <https://youtu.be/ySN5Wnu88nE?si=pqNmoniOccJ8SvyF>)

Ketiga algoritma akan berusaha untuk mencari rute dari node S (start) ke node E (end). Kemudian akan ditranslasikan juga dalam implementasi untuk memecahkan permasalahan word ladder. Setiap algoritma akan memiliki Langkah yang serupa, dengan nilai evaluasi yang berbeda.

Setiap algoritma dimulai dengan pembuatan penyimpanan open node dan close node. Dalam program open node akan dibilang sebagai `active_nodes` dan close nodes adalah `visited`. Penyimpanan `active_nodes` akan memiliki tipe data struktur `priority queue` berdasarkan nilai evaluasi dari nodenya. Sementara `visited` akan berupa himpunan. Kemudian tersedia juga `parentMap` yang berfungsi untuk menyimpan orangtua dari node pada key berupa kata node mereka.

Setelah ketiga data struktur tersebut dibentuk, Baru dapat melakukan traversal graf. Pertama yang dilakukan adalah memasukkan node kata awal dan nilai evaluasinya ke dalam `active_nodes`. Kemudian, node terdepan dari `active_nodes` akan diambil sebagai node saat itu dan mencocokkan node tersebut dengan tujuannya. Jika mengirim false, akan node yang diambil akan disimpan pada `visited`. Lalu setiap node yang bersebelahan akan dimasukkan

ke dalam `active_nodes` dan akan memasukkan data baru pada `parentMap` untuk mengetahui asal nodenya.

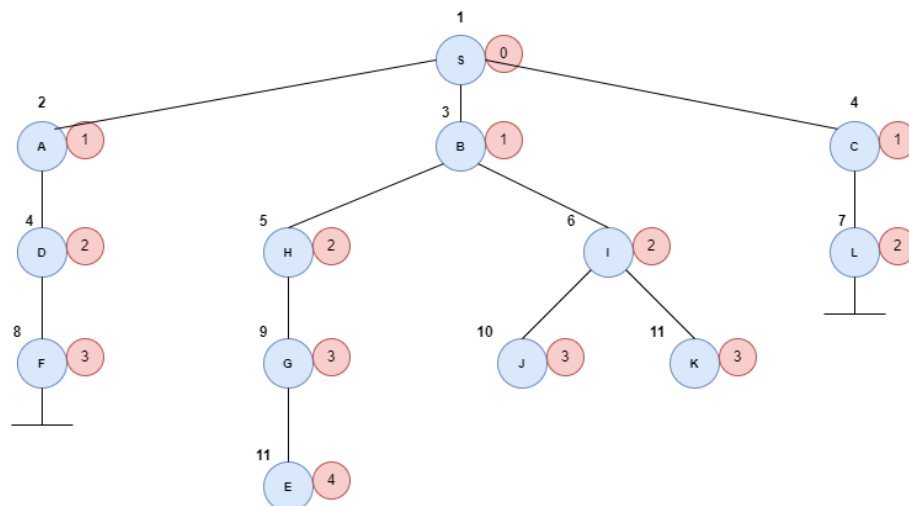
Lalu akan terjadi pengulangan berupa ambil node terdepan dari `active_nodes`, mencocokkan dengan tujuan, jika masih salah masukkan ke himpunan `visited` dan telusuri node yang bersebelahan dengan node tersebut, dan sekali lagi memasukkan node-node tersebut dalam `active_nodes` dan `parentMap` jika node masih belum dijelajahi. Program akan terus mengulang algoritma tersebut sampai node yang terambil adalah node tujuan atau `active_nodes` tidak memiliki node lagi.

Jika mendapatkan solusinya akan dilakukan konstruksi ulang dari node akhir ke node asalnya hingga tidak terdapat lagi key dalam `parentMap`. Dalam implementasinya, akan digunakan linked list, supaya pemasukkan node di depan akan memiliki kompleksitas $O(1)$. Setelah itu program akan menunjukkan jumlah node yang dilalui, jalur dari word ladder tersebut, dan waktu yang dibutuhkan oleh algoritma.

3.1.1 Nilai Evaluasi Algoritma Uniform-Cost Search

Pada algoritma ini yang telah secara singkat dijelaskan pada bagian sebelumnya, UCS bersifat uninformed. Maka nilai evaluasi didapat dari jarak dari node awal ke node yang akan dilalui. Dalam -permainan word ladder, setiap kata yang bersebelahan akan memiliki jarak yang sama. Sehingga jarak tersebut bisa ditentukan sebagai 1.

Pada UCS sesungguhnya, setelah terdapat node yang merupakan tujuan program akan terus berlanjut untuk mencari lagi solusi dengan jarak yang lebih rendah, karena mungkin saja jarak antara node solusi menjauhkan dari tujuan walaupun node bersebelahan. Jika node yang diambil dari `active_nodes` memiliki jarak yang lebih jauh (cost lebih tinggi), data itu algoritma akan berhenti. Maka seharusnya, node dalam `active_nodes` akan diperiksa kembali setelah solusi didapat. Namun karena jarak selalu sama dengan kedalaman, setelah mendapat solusi, dapat langsung dihentikan. Berikut adalah ilustrasi urutan penelusurannya.

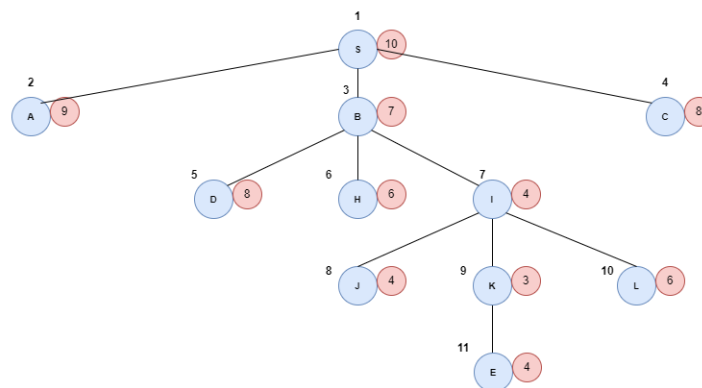


Gambar 3.1.1.1 Ilustrasi Urutan Pencarian UCS
(Sumber: Direktori Lokal)

Dilihat dari ilustrasi, urutan dari penarian node sama seperti algoritma BFS. Ini dikarenakan jarak yang memiliki nilai yang sama antar node. Sehingga cost dengan kedalaman akan memiliki nilai yang sama. Dapat dilihat juga, pada algoritma ini dilakukan penelusuran sebanyak 13 kali (Terdapat duplikat untuk 4 dan 11).

3.1.2 Nilai Evaluasi Algoritma Greedy Best First Search

Algoritma ini bersifat heuristic sehingga memiliki suatu fungsi evaluasi khusus. Sebagai contohnya adalah *hamming distance* yang berupa jarak garis lurus dari suatu node ke node tujuan. Berikut adalah ilustrasi dari GBFS dengan hamming distance yang didapat dari pranala sumber Gambar 3.1.1.



Gambar 3.1.2.1 Ilustarsi Urutan Pencarian GBFS
(Sumber: Direktori Lokal)

Jika dibandingkan dengan algoritma UCS, GBFS menghabiskan ruang yang lebih sedikit. Kemudian karena node yang diekspansi lebih sedikit, akan menghabiskan waktu yang lebih sebentar juga. Namun solusi yang didapat GBFS dapat memiliki jarak yang tidak optimal, karena dapat terjadi node yang bersebelahan memiliki nilai fungsi yang relatif sama terhadap tujuan, atau menjauhi terlebih dahulu walaupun dalam ilustrasi tidak terlihat.

Dalam implemntasi untuk word ladder, nilai hasil evaluasi didapat dengan menghtiung jumlah huruf yang berbeda antara node tersebut dengan tujuannya. Fungsi heuristic yang baik adalah fungsi yang memiliki nilai lebih kecil dibanding jarak sesungguhnya. Fungsi tersebut dapat dibuktikan kurang, karena jumlah perbedaan huruf adalah total Langkah minimum yang mungkin dari suatu pasangan kata. Berikut adalah contoh dari hasil fungsinya.



Gambar 3.1.2.2 Contoh Dua Kata
(Sumber: <https://wordwormdormdork.com>)

Melalui fungsi yang telah ditentukan, maka evaluasi nilai menjadi 2. Maka nilai tersebut akan menjadi cost yang dimiliki oleh node 'SPITE'. Kemudian perlu diberikan peringatan, implementasi dalam source code, mungkin terjadi backtrack, untuk mencari melalui node aktif yang memiliki cost terkecil.

3.1.3 Nilai Evaluasi Algoritma A*

Algoritma ini akan menggunakan kedua nilai dari UCS dan GBFS sebagai cost dari node. Dengan menggabungkannya, maka algoritma ini dapat mencari solusi yang optimal dengan waktu yang lebih sedikit dibandingkan dengan UCS tetapi masih lebih komplet dan optimal dibandingkan dengan GBFS. Maka bisa disimpulkan cost $f(n)$ memiliki rumus:

$$f(n) = g(n) + h(n)$$

Dengan $g(n)$ berupa kedalaman node dari node mulai dan $h(n)$ merupakan estimasi jarak dari node tersebut ke tujuan. Secara teoritis, algoritma ini akan membutuhkan memori yang serupa juga dengan kecepatannya, lebih hemat daripada UCS tetapi lebih boros dibanding GBFS.

3.2 Source Code

3.2.1 englishWords.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.*;

public class englishWords {
    static Map<Integer, Set<String>> english_words = new HashMap<>();

    static {
        load_sowpods();
    }

    public static void load_sowpods() {
        String filename = "sowpods.txt";
        String filedir = "english_words";
        try {
            BufferedReader english_file = new BufferedReader(new
FileReader("./" + filedir + "/" + filename));
            String word;
            while ((word = english_file.readLine()) != null) {
                // System.out.println(word);
                english_words.computeIfAbsent(word.length(), k -> new
HashSet<String>()).add(word);
            }
            english_file.close();
        } catch (FileNotFoundException FNFe){
```



```

        FNFe.printStackTrace();
    } catch(IOException IOe) {
        IOe.printStackTrace();
    }
}

public static Set<String> getDictionary(int word_length) {
    return english_words.get(word_length);
}

public static void printDictionary() {
    english_words.forEach((key, values) -> {
        values.forEach(value -> {System.out.println("Key: " + key);
System.out.println("Value: " + value);});
    });
}

// Calculate the difference between 2 word (For heuristic function)
public static int charDiff(String word_1, String word_2) {
    if (word_1.length() != word_2.length()) {
        System.out.println("\n The strings are not of the same length");
        return -1;
    }
    int diff_counter = 0;
    for (int i = 0; i < word_1.length(); i++) {
        if (word_1.charAt(i) != word_2.charAt(i)) {
            diff_counter++;
        }
    }
    return diff_counter;
}

public static boolean validWords(String start, String end) {
    if (start.length() != end.length() || start.length() > 15 ||
start.length() < 1 || end.length() > 15) {
        return false;
    }
    return (englishWords.getDictionary(start.length()).contains(start) &&
englishWords.getDictionary(end.length()).contains(end));
}
}

```

Kelas ini digunakan sebagai tempat membaca seluruh kata-kata yang tersedia dalam sowpods.txt. Terdapat map English_words dengan panjang kata sebagai key ke set yang berisi kata-kata dengan Panjang tersebut, supaya saat mencari solusi, tidak harus menelusuri seluruh kata.

3.2.2 treeNode.java

```

public class treeNode {
    String word;
    int cost;

    treeNode(String word, int cost) {
        this.word = word;
        this.cost = cost;
    }
}

```

Kelas ini digunakan untuk menyimpan node yang berisi kata dan cost yang merupakan nilai dari $f(n)$ masing-masing algoritma.

3.2.3 solver.java

```

import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;

abstract class solver {
    protected String start;
    protected String end;
    protected Set<String> used_dictionary;
    protected List<String> path_result;

    public void findSolution() {}

    public void printResult() {
        if (path_result == null) {
            System.out.println(" There are no path found from " + start + " to " + end);
            return;
        }
        System.out.println("Here is the path:");
        int i = 0;
        for (; i < path_result.size() - 1; i++) {
            System.out.printf("%s -> ", path_result.get(i));
        }
        System.out.println(path_result.get(i));
    }

    protected void constructPath (Map<String, String> parentMap) {
        List<String> path = new LinkedList<>();
        String currentWord = this.end;

        while (currentWord != null) {
            path.add(0, currentWord);
            currentWord = parentMap.get(currentWord);
        }
    }
}

```

```

    }

    this.path_result = path;
}
}

```

Sebagai abstrak kelas algoritma. Memiliki atribut kata awal, kata tujuan, kamus kata yang digunakan, dan hasil dari pencarian solusi. Seluruh kelas algoritma akan menjadi subclass dari kelas ini dan melakukan override terhadap fungsi findSolution. Memiliki juga printResult untuk menunjukkan hasil dan constructPath untuk membangun ulang solusi.

3.2.4 UCS.java

```

import java.util.*;

public class UCS extends solver{
    public UCS(String startingWord, String goalWord) {
        this.start = startingWord;
        this.end = goalWord;
        this.used_dictionary =
englishWords.getDictionary(this.start.length());
    }

    public void findSolution() {
        PriorityQueue<TreeNode> active_nodes = new
PriorityQueue<>(Comparator.comparingInt(node -> node.cost));

        Set<String> visited = new HashSet<>();

        Map<String, String> parentMap = new HashMap<>();

        active_nodes.add(new treeNode(start, 0));

        while (!active_nodes.isEmpty()) {
            treeNode currentNode = active_nodes.poll();
            String currentWord = currentNode.word;
            int currentCost = currentNode.cost;

            if (currentWord.equals(end)) {
                constructPath(parentMap);
                System.out.println("Node yang dikunjungi : " +
visited.size());
                return;
            }

            if (visited.contains(currentWord)) {
                continue;
            }
        }
    }
}

```

```

        visited.add(currentWord);

        List<String> adjacent = new ArrayList<>();
        for (String candidate : this.used_dictionary) {
            if (englishWords.charDiff(currentWord, candidate) == 1) {
                adjacent.add(candidate);
            }
        }

        for (String nextWord : adjacent) {
            int newCost = currentCost + 1;
            if (!visited.contains(nextWord) &&
!parentMap.containsKey(nextWord)) {
                active_nodes.add(new treeNode(nextWord, newCost));
                parentMap.put(nextWord, currentWord);
            }
        }
    }

    System.out.println(" No solution found...");
}

```

Kelas untuk algoritma uniform-cost search. Melakukan extends terhadap kelas abstrak solver. Node diisi dengan cost berupa jarak node tersebut dari start. Dalam implementasinya, karena jarak seluruh word dan hubungannya hanya satu, maka jarak sama saja dengan kedalamannya. Algoritma ini mirip dengan algoritma BFS, karena pada dasarnya UCS akan selalu menelusuri semua node pada kedalaman tertentu dahulu, sehingga solusi pertama yang ditemukan, pasti juga solusi paling optimal.

3.2.5 GBFS.java

```

import java.util.*;

public class GBFS extends solver{
    public GBFS(String startingWord, String goalWord) {
        this.start = startingWord;
        this.end = goalWord;
        this.used_dictionary =
englishWords.getDictionary(this.start.length());
    }

    @Override
    public void findSolution(){
        PriorityQueue<treeNode> active_nodes = new
PriorityQueue<>(Comparator.comparingInt(node -> node.cost));

        Set<String> visited = new HashSet<>();
    }
}

```

```

        Map<String, String> parentMap = new HashMap<>();

        active_nodes.add(new treeNode(start, englishWords.charDiff(start,
end)));

        while (!active_nodes.isEmpty()){
            treeNode currentNode = active_nodes.poll();
            String currentWord = currentNode.word;

            if (currentWord.equals(end)) {
                constructPath(parentMap);
                System.out.println("Node yang dikunjungi : " +
visited.size());
                return;
            }

            if (visited.contains(currentWord)) {
                continue;
            }

            visited.add(currentWord);

            List<String> adjacent = new ArrayList<>();
            for (String candidate : this.used_dictionary) {
                if (englishWords.charDiff(currentWord, candidate) == 1) {
                    adjacent.add(candidate);
                }
            }

            for (String nextWord : adjacent) {
                if (!visited.contains(nextWord) &&
!parentMap.containsKey(nextWord)) {
                    active_nodes.add(new treeNode(nextWord,
englishWords.charDiff(nextWord, end)));
                    parentMap.put(nextWord, currentWord);
                }
            }
        }

        System.out.println(" No solution found...");
    }
}

```

Kelas untuk algoritma greedy best first search. Melakukan extends terhadap kelas abstrak solver dan melakukan override terhadap fungsi findSolution. Node diisi dengan cost berupa estimasi jarak node ke tujuannya berdasarkan *hamming distance* yang berupa banyaknya perbedaan karakter di antara kata tersebut. Perlu diberi peringatan, dalam

implementasinya, akan melakukan ekspansi dengan cost terendah sehingga backtrack mungkin terjadi.

3.2.6 Source Code A*

```
import java.util.*;

class ASTAR extends solver {
    public ASTAR(String startingWord, String goalWord) {
        this.start = startingWord;
        this.end = goalWord;
        this.used_dictionary =
englishWords.getDictionary(this.start.length());
    }

    @Override
    public void findSolution() {
        PriorityQueue<treeNode> active_nodes = new
PriorityQueue<>(Comparator.comparingInt(node -> node.cost));

        Set<String> visited = new HashSet<>();

        Map<String, String> parentMap = new HashMap<>();

        active_nodes.add(new treeNode(start, englishWords.charDiff(start,
end)));

        while (!active_nodes.isEmpty()){
            treeNode currentNode = active_nodes.poll();
            String currentWord = currentNode.word;
            int distance_from_start = currentNode.cost -
englishWords.charDiff(currentWord, end);

            if (currentWord.equals(end)) {
                constructPath(parentMap);
                System.out.println("Node yang dilewati : " + visited.size());
                return;
            }

            if (visited.contains(currentWord)) {
                continue;
            }

            visited.add(currentWord);

            List<String> adjacent = new ArrayList<>();
            for (String candidate : this.used_dictionary) {
                if (englishWords.charDiff(currentWord, candidate) == 1) {
                    adjacent.add(candidate);
                }
            }
        }
    }
}
```

```

    }
}

for (String nextWord : adjacent) {
    int new_cost = distance_from_start + 1 +
englishWords.charDiff(nextWord, end);
    if (!visited.contains(nextWord) &&
!parentMap.containsKey(nextWord)) {
        active_nodes.add(new treeNode(nextWord, new_cost));
        parentMap.put(nextWord, currentWord);
    }
}
}

System.out.println(" No solution found...");
}
}

```

Kelas untuk algoritma A* Melakukan extends terhadap kelas abstrak solver dan melakukan override terhadap fungsi findSolution. Node diisi dengan cost berupa estimasi jarak node ke tujuannya berdasarkan banyaknya karakter yang berbeda.

3.2.7 wordladder.java

```

import java.util.InputMismatchException;
import java.util.Scanner;

public class wordladder {
    static Scanner sc = new Scanner(System.in);
    public static void main(String args[]) {
        printWelcome();
        System.out.print(" Start word: ");
        String start_string = sc.nextLine().trim().toUpperCase();
        System.out.print(" Target word: ");
        String end_string = sc.nextLine().trim().toUpperCase();

        long startTime = System.currentTimeMillis();
        if (englishWords.validWords(start_string, end_string)) {
            int method_select = methodSelector();
            startTime = System.currentTimeMillis();
            solver wordLadderSolver;
            switch (method_select) {
                case 1:
                    System.out.println("+-----+");
                    System.out.println("| Solving using Uniform-Cost
Search |");
                    System.out.println("+-----+");
                    wordLadderSolver = new UCS(start_string, end_string);

```

```

        break;
    case 2:
        System.out.println("+-----+");
        System.out.println("| Solving using Greedy Best First
Search      |");
        System.out.println("+-----+");
        wordLadderSolver = new GBFS(start_string, end_string);
        break;
    default:
        System.out.println("+-----+");
        System.out.println("| Solving using
A*          |");
        System.out.println("+-----+");
        wordLadderSolver = new ASTAR(start_string, end_string);
        break;
    }
    wordLadderSolver.findSolution();
    wordLadderSolver.printResult();
} else if (start_string.equals("CATHY") || end_string.equals("CATHY"))
{
    System.out.println(" Who is *****?");
} else if (start_string.equals("CATHERINE") ||
end_string.equals("CATHERINE")) {
    System.out.println(" Who is *****?");
}
else {
    System.out.println(" The strings you have chosen are not
eligible");
}
long elapsed_time = System.currentTimeMillis() - startTime;
System.out.println("\n</> The program took " + elapsed_time + " ms");
sc.close();
}

private static void printWelcome() {
    System.out.println("+-----+");
    System.out.println("| Welcome to word ladder solver (Using SOWPODS
dictionary) |");
    System.out.println("+-----+");
}

private static void printSelection() {

```



```

        System.out.println("+----- SELECT METHODS -----
-----+");
        System.out.println("| 1. Uniform-Cost Search
algorithm          |");
        System.out.println("| 2. Greedy Best First Search
algorithm          |");
        System.out.println("| 3. A*
algorithm          |");
        System.out.println("+-----
-----+");
    }

    private static int methodSelector() {
        int input = 0;

        printSelection();

        while (input < 1 || input > 3) {
            System.out.print(" Please select the method's number: ");
            try {
                input = sc.nextInt();
                if (input < 1 || input > 3) {
                    System.out.println("\n There's no such option, please try
again!\n");
                }
            } catch (InputMismatchException e) {
                System.out.println("\n Please input an integer!\n");
                sc.next();
            }
        }

        return input;
    }
}

```

Kelas utama untuk menjalankan program. Menerima input dan sebagainya, walaupun tidak menyimpan data apa-apa. Pada fungsi main terdapat switch yang membuat kelas dari salah satu jenis algoritma. Terdapat juga fungsi untuk memperindah CLI.

BAB 4 UJI COBA

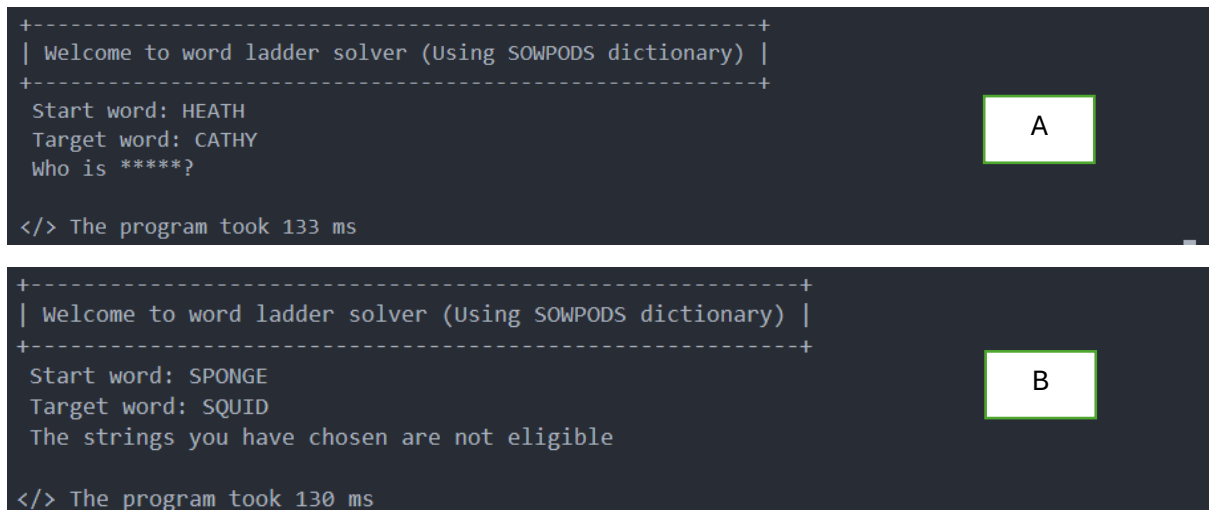
4.1 Tata Cara Penggunaan Program

Program memerlukan Bahasa pemrograman java (Penulis menggunakan java 17.0.8).

Berikut cara untuk memulai menggunakan program:

1. Buka CMD dan set direktori ke './Tucil3_13522160/'
2. Masukkan command berikut pada CMD, "java -jar .\bin\Tucil3_13522160.jar"
(alternatif)
Masukkan command, "javac -d bin src/*.java"
Kemudian masukkan command, "java -cp bin wordladder"
3. Program akan dimulai
4. Masukkan kata awal dan tujuan
5. Masukkan input angka yang merujuk pada metode yang digunakan
(alternatif)
Jika input kata salah karena tidak ada di kamus ataupun panjang yang berbeda, program akan langsung keluar setelah menunjukkan waktu eksekusi program
6. Setelah program dijalankan, akan ditampilkan jumlah node yang dilewati, solusinya, dan waktu yang dibutuhkan program
(alternatif)
Tidak akan menunjukkan node yang dilalui dan solusinya

4.2 Hasil Uji Coba



```
+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: HEATH
Target word: CATHY
Who is *****?

</> The program took 133 ms
```

```
+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: SPONGE
Target word: SQUID
The strings you have chosen are not eligible

</> The program took 130 ms
```

Gambar 4.2.1 Uji Coba Kata Invalid; A: Tidak di kamus (Clear All Cathy); B: Panjang berbeda)

(Sumber: Direktori Lokal)


```

+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: APPLE
Target word: GUAVA
+----- SELECT METHODS -----+
| 1. Uniform-Cost Search algorithm |
| 2. Greedy Best First Search algorithm |
| 3. A* algorithm |
+-----+
Please select the method's number: 1
+-----+
| Solving using Uniform-Cost Search |
+-----+
Node yang dikunjungi : 6865
Here is the path:
APPLE -> AMPLE -> AMOLE -> ANOLE -> ANILE -> ANISE -> ARISE -> GRISE -> GRIME -> GRAME -> GRAMA -> GRANA -> GUANA -> GUAVA
</> The program took 2501 ms

```

Gambar 4.2.4 Uji Coba Apple ke Guava UCS
(Sumber: Direktori Lokal)

```

+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: SKEDADDLED
Target word: SKEDADDLER
+----- SELECT METHODS -----+
| 1. Uniform-Cost Search algorithm |
| 2. Greedy Best First Search algorithm |
| 3. A* algorithm |
+-----+
Please select the method's number: 3
+-----+
| Solving using A* |
+-----+
Node yang dilewati : 1
Here is the path:
SKEDADDLED -> SKEDADDLER
</> The program took 17 ms

```

Gambar 4.2.5 Uji Coba Skedaddled ke Skedaddler GBFS
(Sumber: Direktori Lokal)

```

+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: PAWN
Target word: ROOK
+----- SELECT METHODS -----+
| 1. Uniform-Cost Search algorithm |
| 2. Greedy Best First Search algorithm |
| 3. A* algorithm |
+-----+
Please select the method's number: 2
+-----+
| Solving using Greedy Best First Search |
+-----+
Node yang dikunjungi : 5
Here is the path:
PAWN -> POWN -> POON -> POOK -> ROOK
</> The program took 10 ms

```

Gambar 4.2.6 Uji Coba Pawn ke Rook A*
(Sumber: Direktori Lokal)

```

+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: CLOAK
Target word: FRIES
+-----+
| 1. Uniform-Cost Search algorithm |
| 2. Greedy Best First Search algorithm |
| 3. A* algorithm |
+-----+
Please select the method's number: 1
+-----+
| Solving using Uniform-Cost Search |
+-----+
Node yang dikunjungi : 2433
Here is the path:
CLOAK -> CROAK -> CREAK -> CREEK -> CREES -> CRIES -> FRIES
</> The program took 920 ms

```

Gambar 4.2.7 Uji Coba Cloak ke Fries UCS
(Sumber: Direktori Lokal)

```

+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: BIRDS
Target word: SLING
+-----+
| 1. Uniform-Cost Search algorithm |
| 2. Greedy Best First Search algorithm |
| 3. A* algorithm |
+-----+
Please select the method's number: 3
+-----+
| Solving using A* |
+-----+
Node yang dilewati : 699
Here is the path:
BIRDS -> BARDS -> SARDS -> SAIDS -> SAINS -> SPINS -> SPINK -> SLINK -> SLING
</> The program took 320 ms

```

Gambar 4.2.8 Uji Coba Birds ke Sling GBFS
(Sumber: Direktori Lokal)

```

+-----+
| Welcome to word ladder solver (Using SOWPODS dictionary) |
+-----+
Start word: SMART
Target word: PHONE
+-----+
| 1. Uniform-Cost Search algorithm |
| 2. Greedy Best First Search algorithm |
| 3. A* algorithm |
+-----+
Please select the method's number: 2
+-----+
| Solving using Greedy Best First Search |
+-----+
Node yang dikunjungi : 6
Here is the path:
SMART -> SCART -> SCARE -> SCORE -> SHORE -> SHONE -> PHONE
</> The program took 19 ms

```

Gambar 4.2.9 Uji Coba Smart ke Phone A*
(Sumber: Direktori Lokal)

4.3 Analisis Uji Coba

Berdasarkan uji coba, dapat terbukti hipotesis teori benar. Dari jumlah node yang dikunjungi dan total waktu yang dihabiskan, GBFS paling cepat dan sedikit memori disusul oleh A* dan kemudian UCS. Tetapi GBFS dalam Gambar 4.2.3, memiliki solusi dengan

jarak yang lebih Panjang, sedangkan kedua algoritma lainnya, adalah solusi paling optimal berdasarkan kamus SOWPODS.

BAB 5 PENUTUP

5.1 Kesimpulan

Ketiga algoritma yang digunakan merupakan modifikasi dari algoritma Dijkstra dalam mencari rute dalam suatu graf. Ketiga algoritma dapat memiliki langkah yang serupa dengan parameter yang berbeda untuk menilai masing-masing node. Heuristik digunakan untuk GBFS dan A* membuat program memiliki potensial waktu yang lebih singkat dan memori yang tidak seboros UCS. Kembali lagi terhadap permasalahan performa dan solusi yang optimal, algoritma A* adalah algoritma yang paling cocok di antara yang lainnya dalam memecahkan word ladder.

5.2 Saran

Jika ingin membuat algoritma GBFS, ditemukan dua versi, bisa backtrack dan tidak bisa backtrack. Dalam program yang dibuat, dapat melakukan backtrack. Selain itu, kamus Bahasa Inggris yang digunakan dapat beragam. Pada tahap penelitian, ditemukan kamus seperti words-alpha, Collins, OWTCL, OWL, dan yang digunakan SOWPODS. Dalam situs word ladder yang dilampirkan pada spesifikasi tugas, tidak menggunakan kamus SOWPODS, tetapi mungkin OWL atau OWTCL yang hak ciptanya dimiliki NASPA, sehingga sulit untuk mendapat kamusnya kecuali dengan zyzyva.

DAFTAR PUSTAKA

- Munir, Rinaldi. 2020. “Penentuan rute (Route/Path Planning) - Bagian 1”. Diakses pada 5 Mei 2024. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>
- Munir, Rinaldi. 2020. “Penentuan rute (Route/Path Planning) - Bagian 1”. Diakses pada 5 Mei 2024. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>

LAMPIRAN

Pranala repository GitHub:

https://github.com/Rinnearu/Tucil3_13522160

Checklist:

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma UCS	✓	
3. Solusi yang diberikan pada algoritma UCS optimal	✓	
4. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma Greedy Best First Search	✓	
5. Program dapat menemukan rangkaian kata dari start word ke end word sesuai aturan permainan dengan algoritma A*	✓	
6. Solusi yang diberikan pada algoritma A* optimal	✓	
7. [Bonus] : Program memiliki tampilan GUI		✓